TP C++ :Héritage

B 3231 MEYER Franck, ROYER Robin 4 Février 2015

[franck.meyer@insa-lyon.fr](mailto:franck.meyer@insa-lyon.fr)

[robin.royer974@insa-lyon.fr](mailto:robin.royer974@insa-lyon.fr)

# Conception

STRUCTURE DE DONNEES :

Afin de stocker les objets de notre modèle courant, nous allons utiliser 2 maps dans notre classe ObjectManager, une pour stocker toutes les classes filles de Shape correspondant à chaque type d’objet et pouvant être enregistré et une pour stocker les sélections

GESTION DU PRINCIPE UNDO/REDO :

Afin de gérer l’historique des actions ainsi que les UNDO/REDO, nous avons utilisé un « design pattern » History et des classes filles propres à chaque type d’action.

Elles sont stockées dans un deque qui gère un historique limité aux 20 dernières commandes. Un attribut entier permet de localiser dans le deque l’action en cours. Chacune d’entre elle peut annuler ou répéter l’action qui lui est propre

UML :

# Spécifications

## classe ObjectManager :

La classe ObjectManager gère les objets du modèle courant grâce à une première map de pointeur de « Shape », les sélections du modèle grâce à une seconde map de pointeur de « Selection », l’historique des actions effectuées sur le modèle grâce un deque de pointeur « History » et enfin en entier non signé qui localise la commande actuelle dans le deque en fonction des « UNDO/REDO ».

addObject

AddObject prend en paramètre un pointeur de « Shape » et renvoit un booléen true si l’objet pointé a été bien ajouté et false si l’objet n’a pas été ajouté (nom déjà utilisé).

addSelection

AddSelection prend en paramètre un pointeur de « selection » et renvoit un booléen true si la sélection a bien été inséré et false si la sélection n’a pas été ajouté (nom déjà utilisé).

getSelectedShapes

GetSelectedShapes prend en paramètre 4 entiers « long » correspondant aux coordonnées de 2 points et renvoit un vector de pointeur de « Shape » pointant sur toutes les « Shapes » contenues dans le rectangle délimité par les 2 points.

## contains

Contains prend en paramètre une chaine de caractères correspondant à un nom d’objet et renvoit un booléen true si le nom correspondant est dans la première ou deuxième map et false dans le cas contraire

containsNames

ContainsNames prend en paramètre un vector de chaines de caractères corespond à des noms d’objets et renvoit un booléen true si tout les noms sont présent dans la structure de données et false si l’un d’eux n’y est pas.

removeObjects

RemoveObjects prend en paramètre un vector de chaine de caractères correspondant aux noms des objets à supprimer et renvoit un vector de pointeurs de « Shape » des objets qui ont été supprimés du modèle.

## moveObject

MoveObject prend en paramètre une chaine de caractère correspondant à l’objet ou la selection à bouger, 2 entiers longs correspondant aux déplacements selon la coordonées X et la coordonnées Y et renvoit un vector de pointeur de « Shape » pointant sur les « Shapes » déplacées.

## printDescs

PrintDescs affiche sur le flux de sortie la description des « shapes » contenues dans la première map selon la même descritption que l’entrée en fonction du type de « Shapes ».

## clearObjects

ClearObjects ne prend pas de paramètres et renvoit un vector de pointeur de « shapes » pointant sur tout les objets qui vont être supprimés et retire tout les objets de la structure de données.

## addtohistory

AddToHistory prend en paramètre un pointeur « History », est de type void, ajoute à l’historique le paramètre en entrée, supprime la partie de l’historique d’index supérieur à l’attribut index et supprime le bas de l’historique quand celui-ci dépasse une taille de 20.

## undocurrent

UndoCurrent ne prend pas de paramètre, retourne un booléen false si l’historique est vide sinon renvoie true et si la commande correspondant à l’index est annulé et l’index correspond alors la commande précedente.

## redocurrent

RedoCurrent ne prend pas de paramètre, retourne un booléen false si il n’y a pas de commande à répéter sinon répéter les commande annuler une à une.

## classe Shape :

La classe Shape est une classe abstraite dont vont hériter les objets et sélections de l’application et ayant comme attributs une chaine de caractères correspondant au nom de la forme et un booléen isRemoved pour connaitre l’état de la forme.

## getdescriptor

Méthode virtuel pure sans paramètre, redéfinie dans les classes filles afin de renvoyer la description de chaque objet sous la forme d’ une chaine de caractères identique au format d’entrèe.

## remove

Remove est une méthode virtuel qui ne prend pas de paramètre, ne renvoit rien et fait passer l’attribut isRemoved de l’objet courant à false afin de le considérer supprimé.

## remake

Remake est une méthode virtuel qui ne prend pas de paramètre, ne renvoit rien et fait passer l’attribut isRemoved de l’objet courant à true afin de le considérer de nouveau actif.

## move

Méthode virtuel pure prenant en paramètres 2 entiers long correspondant aux déplacements selon X et Y et renvoyant un vector de pointeurs de « Shape ». Cette méthode est redéfinit dans les classes filles afin d’appliquer le déplacements à chaque type d’objet.

## isencompassed

Méthode virtuelle pure prenant en paramètre 4 entiers longs correspondants aux coordonnées de 2points et renvoit un booléen. Elle est redéfinit dans les classes filles et renvoit true si l’objet en question est compris dans le rectangle définit par les 2 points et false dans le cas contraire.

## getname

GetName est un accesseur retournant en chaine de caractères l’attribut name.

## isdeleted

isDeleted est un accesseur retournant l’attribut booléen isRemoved.

## Classe History :

La classe History est une classe abstraite dont vont hériter les différentes instances d’action à « sauvegarder »

## UNDO

Méthode virtuelle pure prenant un pointeur sur l’ObjectManager de type void qui sera redéfinie par chaque instance d’action à sauvegarder, permettant d’annuler l’action correspondante.

## redo

Méthode virtuelle pure prenant un pointeur sur l’ObjectManager de type void qui sera redéfinie par chaque instance d’action à sauvegarder permettant de répéter l’action correspondante.

# COMMANDES

Le contrôle de l’éditeur de formes géométriques par l’utilisateur se fera en mode console (les entrées, sorties mais aussi validation et messages d’erreur).

Il y a 2 types de commandes :

* Les commandes qui agissent sur le modèle, en modifiant les objets le composant
* Les commandes de gestion de modèle: l’énumération des objets du modèle, la sauvegarde du modèle etc.

Les lignes commençant par le caractère dièse sont des commentaires et ne sont pas pris en compte.

Le nom des objets est composé de lettres et de chiffres.

Commandes possibles :

Ajout d’un cercle: C Name X Y R

Ajout d’un rectangle: R Name X1 Y1 X2 Y2

Ajout d’une Ligne: L Name X1 Y1 X2 Y2

Ajout d’une poly-ligne: PL Name X1 Y1 X2 Y2 … Xn Yn

Création d’une sélection: S Name X1 Y1 X2 Y2

Suppression: DELETE Name1 Name2 … Name3

Déplacement: MOVE Name dX dY

Enumeration: LIST

Annuler: UNDO

Répéter : REDO

Charger: LOAD FILENAME

Sauvegarder le modèle: SAVE FILENAME

Vider le modèle: CLEAR

Fermer l’application: EXIT

Sortie :

Confirmation : OK

Erreur : ERR

Enumération : Desc1

Desc2

DescN